On-demand Bound Computation for Finding Leading Solutions to Soft Constraints

Martin Sachenbacher and Brian C. Williams
MIT Computer Science and AI Laboratory
September 27, 2004

**Finding Leading Solutions**

- Many AI problems = Constraint optimization problems
  - Diagnosis (state estimation)
  - Planning
  - ...
- Practical AI requirement: Robustness ⇒ Generate solutions in best-first order, until halted
  - Most likely diagnoses, until failure is found
  - Least cost plans, until actions are executable
  - ...
- Problem: Not known in advance when halted ⇒ Must generate each solution as quickly as possible

**Example: Full Adder Diagnosis**

- Variables \{u, v, w, y, a1, a2, e1, e2, o1\}
- \{a1, a2, e1, e2, o1\} describe modes of gates
- Gates are either in good ("G") or broken ("B") mode

![Full Adder Diagram]

**Modeling the Example as Soft CSP**

<table>
<thead>
<tr>
<th>(f_{a1} : a1 \ y)</th>
<th>(f_{a2} : a2 \ u \ v)</th>
<th>(f_{e1} : e1 \ u \ y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G 0 0</td>
<td>G 0 0</td>
<td>G 0 0</td>
</tr>
<tr>
<td>G 1 1</td>
<td>G 1 1</td>
<td>G 0 1</td>
</tr>
<tr>
<td>B 0 0</td>
<td>B 0 0</td>
<td>B 0 0</td>
</tr>
<tr>
<td>B 0 1</td>
<td>B 0 1</td>
<td>B 0 1</td>
</tr>
<tr>
<td>B 1 0</td>
<td>B 1 0</td>
<td>B 1 0</td>
</tr>
<tr>
<td>B 1 1</td>
<td>B 1 1</td>
<td>B 1 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(f_{o1} : o1 \ v \ w)</th>
<th>(f_{e2} : e2 \ u)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G 0 0</td>
<td>G 0</td>
</tr>
<tr>
<td>G 0 0</td>
<td>B 0</td>
</tr>
<tr>
<td>B 0 1</td>
<td>B 1</td>
</tr>
<tr>
<td>B 1 0</td>
<td>B 1</td>
</tr>
<tr>
<td>B 1 1</td>
<td>B 1</td>
</tr>
</tbody>
</table>

**Leading Solutions for the Example**

- Gate e1 is B (a1=G, a2=G, e1=B, e2=G, o1=G)
- Gate o1 is B (a1=G, a2=G, e1=G, e2=G, o1=B)
- Next best diagnosis involves double fault \(→\) Stop.

![Leading Solution Diagram]
Heuristic (Bound-Guided) Search

- Best-first (A*) search expands node with best $g \times h$, where $g$ is value so far and $h$ is heuristic estimate
- (Kask Dechter AIJ 01): Compute heuristics using bucket trees and dynamic programming

Bucket Tree

- Scheme for one-by-one variable elimination
- Each node $v_i$ defines sub-problem with constraints $\lambda(v_i)$

Dynamic Programming

$\lambda^{(9)} = f_{01} \downarrow w, a$
$\lambda^{(8)} = f_{02} \downarrow v$

$\frac{v_1}{\lambda^{(9)}}: w$
$\frac{v_2}{\lambda^{(8)}}: w$
$\frac{v_3}{\lambda^{(8)}}: v$
$\frac{v_4}{\lambda^{(8)}}: y$
$\frac{v_5}{\lambda^{(9)}}: a_1 f_{01}$
$\frac{v_6}{\lambda^{(8)}}: a_2 f_{02}$
$\frac{v_7}{\lambda^{(8)}}: e_1 f_{01}$
$\frac{v_8}{\lambda^{(8)}}: e_2 f_{02}$

A* Search Tree

Bucket Tree

Evaluate using dynamic programming
- Store constraint $\lambda(v_i)$ at edge to parent

Dynamic Programming

$\lambda^{(9)} = f_{01} \downarrow w, a$
$\lambda^{(8)} = f_{02} \downarrow v$

$\frac{v_1}{\lambda^{(9)}}: w$
$\frac{v_2}{\lambda^{(8)}}: w$
$\frac{v_3}{\lambda^{(8)}}: v$
$\frac{v_4}{\lambda^{(8)}}: y$
$\frac{v_5}{\lambda^{(9)}}: a_1 f_{01}$
$\frac{v_6}{\lambda^{(8)}}: a_2 f_{02}$
$\frac{v_7}{\lambda^{(8)}}: e_1 f_{01}$
$\frac{v_8}{\lambda^{(8)}}: e_2 f_{02}$

$\frac{v_9}{\lambda^{(8)}}: a_1 f_{01}$

$\frac{v_{10}}{\lambda^{(9)}}: a_1 y, w \{ f_{01} \}$
$\frac{v_{11}}{\lambda^{(9)}}: e_1 y, u \{ f_{01} \}$
Bounds from Bucket Trees

- Assign in reverse order using the evaluation function:
  \[
  h^{(0)} = \bigotimes_{j \in \mathbb{C}, \text{child of } v_{p_j}} \mathcal{A}_j \quad s(v_j)
  \]
  \[
  g^{(0)} = \bigotimes_{j \in \mathbb{C}, \text{not in } \mathcal{A}_j} \frac{\text{Exact Bound}}{v_j \{u, v, w, y, a_1, a_2\} \{f_{a1}, f_{a2}\}}
  \]
  \[
  v_1 \{u, v, w, y, a_1, a_2\} \{f_{a1}, f_{a2}\}
  \]
  \[
  v_2 \{u, v, w, y, e_1, e_2\} \{f_{e1}, f_{e2}\}
  \]
  \[
  v_3 : \{a_1\} \quad f_{a1}
  \]
  \[
  v_4 \{a_1, y, u\} \{f_{a1}\}
  \]
  \[
  v_5 \{e_1, y, u\} \{f_{e1}\}
  \]
  \[
  \text{Tree Decomposition}
  \]
  - Generalization of Bucket Trees
  - Eliminate variables in groups \( p_j \) (partial order)

Bounds from Tree Decompositions

- Assign in reverse order using the evaluation function:
  \[
  h^{(0)} = \bigotimes_{j \in \mathbb{C}, \text{child of } v_{p_j}} \mathcal{A}_j \quad s(v_j)
  \]
  \[
  g^{(0)} = \bigotimes_{j \in \mathbb{C}, \text{not in } \mathcal{A}_j} \frac{\text{Exact Bound}}{v_j \{u, v, w, y, a_1, a_2\} \{f_{a1}, f_{a2}\}}
  \]
  \[
  v_1 \{u, v, w, y, a_1, a_2\} \{f_{a1}, f_{a2}\}
  \]
  \[
  v_2 \{u, v, w, y, e_1, e_2\} \{f_{e1}, f_{e2}\}
  \]
  \[
  v_3 : \{a_1\} \quad f_{a1}
  \]
  \[
  v_4 \{a_1, y, u\} \{f_{a1}\}
  \]
  \[
  v_5 \{e_1, y, u\} \{f_{e1}\}
  \]
**On-Demand Bound Computation**
- If only a few leading solutions are needed, pre-computing $s(v_i)$ becomes inefficient
- Interleave search and dynamic programming to compute $s(v_i)$ only as needed (“on-demand”)

**Dynamic Programming** + **Best-First Search** = **On-demand Bounds**

---

**Preferential Independence**
- For any valuation structure: $a \leq b \Rightarrow c \times a \leq c \times b$
- Sufficient to expand only next best child
- Sufficient to compute $h_i$ only for next best child

---

**Dual Formulation**
- Unary soft constraints (functions as variables)
- Binary hard constraints (equality)
- Compatible order

---

**Approximating H**
- Drop hard (equality) constraints for H
  - Heuristics becomes equal for all children
  - Order of children known if tuples of $f_i$ are sorted
  - Makes it possible to generate $s(v_i)$ only as needed

---

**On-Demand Bound Computation**
- Generate tuples of constraints $s(v_i)$ only as needed

---

**Graphs and Diagrams**

- [Dynamic Programming + Best-First Search = On-demand Bounds]
- [Preferential Independence Diagram]
- [Dual Formulation Diagram]
- [Approximating H Diagram]
- [On-Demand Bound Computation Diagram]
### Experiments

- Random Max-CSPs, first (best) solution

<table>
<thead>
<tr>
<th>T</th>
<th>C</th>
<th>N</th>
<th>K</th>
<th>BFTC (% time)</th>
<th>BFOB (% time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>20</td>
<td>15</td>
<td>4</td>
<td>100%</td>
<td>1.4%</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>15</td>
<td>4</td>
<td>100%</td>
<td>3.2%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>C</th>
<th>N</th>
<th>K</th>
<th>BFTC (% time)</th>
<th>BFOB (% time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>15</td>
<td>10</td>
<td>4</td>
<td>100%</td>
<td>4.5%</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>10</td>
<td>4</td>
<td>100%</td>
<td>14.3%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>C</th>
<th>N</th>
<th>K</th>
<th>BFTC (% time)</th>
<th>BFOB (% time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>20</td>
<td>10</td>
<td>4</td>
<td>100%</td>
<td>9.7%</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>10</td>
<td>4</td>
<td>100%</td>
<td>38.5%</td>
</tr>
</tbody>
</table>

### Related Approach

- BTDval algorithm (Terrioux Jégou CP 03)
- Depth-first branch-and-bound on tree decompositions
- Record tuples for constraints \( s(v_i) \) (“structural goods”)

\[
\begin{align*}
\{u, v, w, y, a, b, c, d\} & \subseteq \{f_{i1}, f_{i2}\} \\
\{a, b\} & \subseteq \{f_{i1}\} \\
\{u, y\} & \subseteq \{f_{i2}\} \\
\{v, w, a\} & \subseteq \{f_{i1}\}
\end{align*}
\]

### Future Work

- Evaluate against depth-first branch-and-bound
  - Best-first (A*) search potentially faster
- Combine with approximate dynamic programming
  - Mini-buckets (Dechter Rish UAI 97)
- Extend to partial orders
  - Semiring-based CSPs (Bistarelli IJCAI 95)
Material

Finding Leading Solutions

% Solutions

Best

2nd
best

3rd
best

...